Loops
Bucles
Continuación

Laboratorio de Programación

Alerta de Spoiler

Cuando se trabaja con bucles, es importante conocer algunas ***palabras clave*** especiales que pueden afectar la forma en que se ejecuta un bucle. ¡Veamos un ejemplo!

El bucle del código se ejecuta hasta que se adivina el 8 o se han realizado 50 intentos para adivinar el número secreto. Esto significa que el ciclo debería dejar de ejecutarse tan pronto como se cumpla una de esas condiciones. Pero algo no parece funcionar, podrias corregirlo?

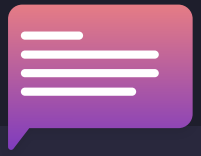
```c
int main() {
    int dato =0; int intento=0;
    while (dato != 8 && intento < 10) {
  printf("Dato incorrecto intenta de nuevo: ");
  scanf("%d", &intento);
  intento++;
}
    return 0;
}
```

Breaking Bad

La palabra clave ***break*** nos permite, literalmente, "salir" de un bucle y evitar que se ejecute más veces.

```c
int main() {
    int intentos=0; int dato=0;
 while (intentos < 10) {
     printf("Ingresa un dato: ");
  scanf("%d", &dato);
  if (dato == 8) {
      printf("HASTA LA VISTA");
    break;
  }
  printf("Dato Incorrecto Intenta de nuevo ");
  intentos++;
}
    return 0;
}
```

Ejercicio 6.1



Realiza un programa para ingresar 10 notas de parciales por teclado, el mismo debe finalizar con la décima nota o mediante la salida ingresando 0. Aplicar el concepto de break visto en el ejemplo del apunte

Continuemos

En un videojuego, luego de una pantalla de "Game Over" a menudo podemos "continuar" desde un guardado o punto de control anterior. Los bucles pueden hacer esto de la misma manera usando continuar, ¡la segunda palabra clave que aprenderemos en esta lección!

En un ciclo, si alguna vez se llega a continuar, se saltará inmediatamente el resto de las declaraciones dentro del cuerpo del ciclo y ***"continuará"*** en la siguiente iteración.

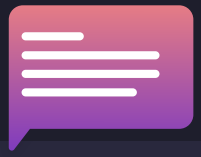
Por ejemplo:

```c
int main() {
for (int i = 0; i < 10; i++) {
  if (i % 2 == 0) {
    continue;
  }
  printf("%d es impar\n", i);
}
    return 0;
}
```

Entonces:

Dentro del ciclo for, vemos una declaración if que verifica si el contador actual, i, es un número par usando el operador %. Si i es par, continúa con la siguiente iteración y omite la declaración de impresión a continuación. Si i es impar, imprime i y continúa normalmente.

Ejercicio 6.2



Realiza un programa que imprima los números de 1 a 99 pero que omita aquellos que son múltiplos de 5. Utiliza la sentencia ***continue;***

Introducción a los Arrays

Laboratorio de Programación

Arrays

Un array es una agrupación de variables del mismo tipo en bloques contiguos de memoria. Esta estructura de datos es especialmente útil en aplicaciones donde hay muchas variables del mismo tipo que están relacionadas entre sí. Considere el ejemplo de tener que almacenar las tres coordenadas de un punto en el espacio (coordenadas x, y y z). Una forma válida de almacenar estas coordenadas en variables es:

```
int xCoord = 1;
int yCoord = 2;
int zCoord = 3;
```

Arrays

Si bien esto está bien, especialmente para una pequeña cantidad de coordenadas, será más difícil de administrar a medida que aumente la cantidad de coordenadas. Una solución a este problema es almacenar las coordenadas en una estructura de datos llamada array de la siguiente manera:

```
int coord[3] = {1, 2, 3};
```

Arrays no inicializados

Cuando creas una array no inicializado, debes especificar su tamaño para que el compilador pueda reservar la cantidad adecuada de bloques de memoria. Una vez que se crea, el tamaño del array no se puede cambiar; esto significa que los array son estáticos. Un array no inicializado se puede llenar más tarde en un programa.

```
int edades[] = {7, 27, 34, 63};
```

O lo que es lo mismo:

```
int edades[4] = {7, 27, 34, 63};
```

Acceso y Modificación de elementos

Se puede acceder a los elementos del array, modificarlos y utilizarlos como cualquier otra variable del mismo tipo de datos.

El primer elemento de un array tiene índice 0 y el último elemento de un array tiene índice tamaño de array - 1. El elemento n está en el índice n-1, por lo que, por ejemplo, el tercer elemento estaría en el índice 2.

Probemos este código

```c
int main() {
 int arr[] = {3, 5, 7, 9};

    arr[2] = 6;
    int x = arr[2];
    printf("%i", arr[4]);

    return 0;
}
```

Recorriendo Arreglos

El principal beneficio de los arreglos es la capacidad de trabajar con grandes cantidades de datos sin asignar a cada dato su propio nombre de variable; por lo tanto, los arreglos suelen contener muchos elementos.

Para trabajar con estos elementos, probablemente tendrás que hacer algo repetidamente.

Como aprendimos anteriormente, se usa un bucle para ejecutar código repetido. Recuerde los dos tipos de bucles: un bucle for y un bucle while.

Recorriendo con while

Primero echemos un vistazo a cómo recorrer un arreglo usando un ciclo while. Considere el ejemplo de tener una arreglo que consta de 20 enteros aleatorios y desea imprimir su contenido en la pantalla. Así es como se hace esto con un bucle while:

```
int arr[] = {6, 9, 18, 37, 4, 23, 27, 16, 1, 30, 22,
    7, 10, 25, 3, 2, 35, 11, 19, 28};
int i = 0;

while(i < 20){
  printf("%i\n", arr[i]);
  i++;
}
```

Ahora con un for

```
int main() {
   int arr[] = {6, 9, 18, 37, 4, 23, 27, 16, 1, 30, 22
       , 7, 10, 25, 3, 2, 35, 11, 19, 28};

  for(int i = 0; i < 20; i++){
    printf("%i\n", arr[i]);
  }
    return 0;
}
```

Ejercicio!

Cree un array y cárgelo con con 100 números múltiplos de 5. Recorra e imprima el array.